MATH60082
Lab Class 5

Contents

5.1 Binomial Tree For Option Pricing

The two most popular models for using binomial trees to price options are

- Cox et al. (1979) (CRR for short) whose extra degree of freedom is to set

$$ud = 1$$

  thus

$$u = e^{\sigma\sqrt{\Delta t}}, \quad d = e^{-\sigma\sqrt{\Delta t}}, \quad q = \frac{e^{r\Delta t} - d}{u - d}$$

- Rendleman and Bartter (1979) who choose:

$$q = \frac{1}{2}$$

  and so

$$u = e^{(r-\frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}}, \quad d = e^{(r-\frac{1}{2}\sigma^2)\Delta t - \sigma\sqrt{\Delta t}}.$$

We wish to generate a stock price tree, so denote the value of the underlying asset after timestep i and upstate j by S_{ij} and we have that:

$$S_{ij} = S_0 u^j d^{i-j}$$

First download and start with this template code:

Click here to download full code

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;
/*
 * Template code for the binomial tree
 */
```

First declare and initialise the Black Scholes parameters for your chosen problem. Here we are going to value a Black Scholes vanilla European call option with, $S_0 = 100$, $X = 100$, $T = 1$, $r = 0.06$ and $\sigma = 0.2$, so declare variables for each of these. Next add in an integer to store the number of steps in the tree and call it n. Finally add in some local variable to describe the tree, so we have the timestep length dt, u, d and q. Your code should look like:-

CLICK HERE TO DOWNLOAD FULL CODE

```
    // declare and initialise Black Scholes parameters
    double S0=100.,X=100.,T=1.,r=0.06,sigma=0.2;
    // declare and initialise tree paramaters (steps in tree)
    int n=3;
    // declare and initialise local variables (u,d,q)
    double dt,u,d,q;
```

Then calculate the values for dt, u, d and q, using the appropriate formula, you should check the value of these against those found in my lecture notes (click here, slide 19).

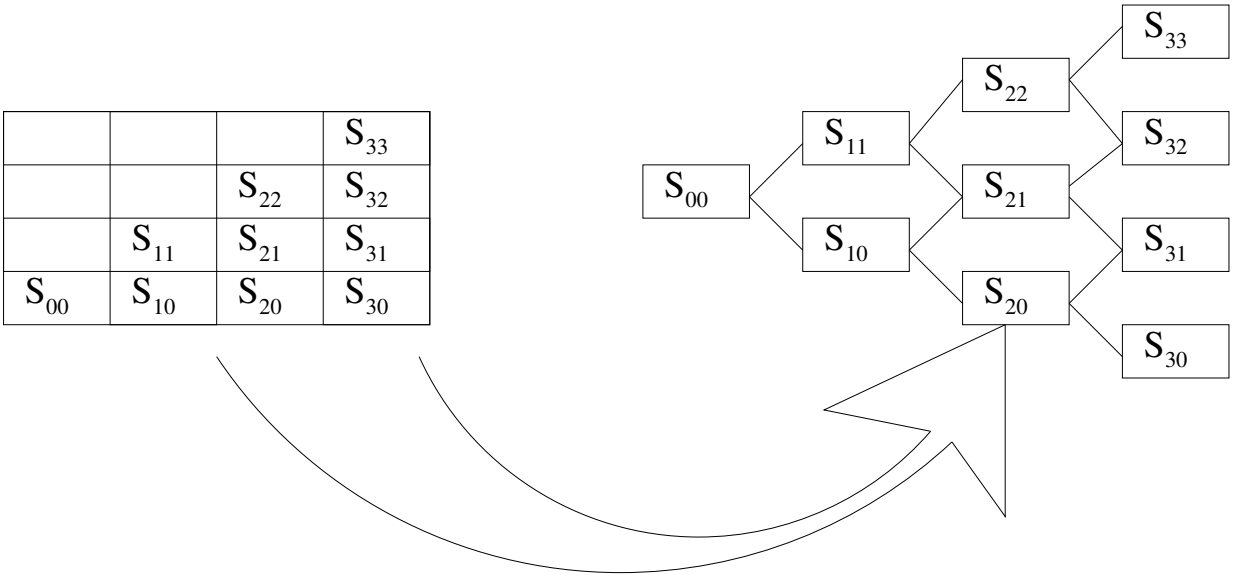
Next we need to create some storage for the values of the stock at each node in the tree. Declare a *vector of vectors* `stockTree`, into which we will place our stock price nodes.

CLICK HERE TO DOWNLOAD FULL CODE

```
    // create storage for the stock price tree and option price tree
    vector<vector<double>> stockTree(n+1,vector<double>(n+1));
```

where this has been intialised to an $n + 1 \times n + 1$ 2D array, and n is the number of nodes in the tree.

Now use a `for` loop and the function `pow` to input the value of the stock at each node in the tree, where $S_{i,j} \rightarrow$`stockTree[i][j]`.



Now print out the vector `stockTree[i][j]` to the screen. Your code might look something like this

CLICK HERE TO DOWNLOAD FULL CODE

```
    // setup and initialise the stock price tree
    for(int i=0;i<=n;i++)
    {
        for(int j=0;j<=i;j++)
        {
            stockTree[i][j]=S0*pow(u,j)*pow(d,i-j);
            cout << i << " " << j << " " << stockTree[i][j] << endl;
        }
        cout << endl;
    }
```

Compare the values with those from the example in the lectures.

5.2 The Option Value Tree

Let us generate a simple example so that we can compare results at every stage to something that we can work out on paper. This is an important idea in debugging, to solve the problem and do all of your bug checking on a small scale **before** attempting the full problem.

First declare a *vector of vectors* which shall hold the values of the option

CLICK HERE TO DOWNLOAD FULL CODE

```
    // create storage for the stock price tree and option price tree
    vector<vector<double>> stockTree(n+1,vector<double>(n+1));
    vector<vector<double>> valueTree(n+1,vector<double>(n+1));
```

and set it to the same size as `stockTree`. Here we use the same relation $V_{i,j} \rightarrow$`valueTree[i][j]`.

Now fill in the final values of the tree, given that we have already first generated the stock tree:

$$V_{n,j} = payoff(S_{n,j}),$$

where payoff is the appropriate function for the type of option we are solving for. For a European call option you should get

CLICK HERE TO DOWNLOAD FULL CODE

```
1     // setup and initialise the final conditions on the option price tree
2     for(int j=0;j<=n;j++)
3     {
4         valueTree[n][j]=max(stockTree[n][j]-X,0.);
5         cout << n << " " << j << " " << valueTree[n][j] << endl;
6     }
```

Now we need to loop **backwards** through the tree to generate the value at each node using the equation:

$$V_{ij} = e^{-r\Delta t}(qV_{i+1,j+1} + (1-q)V_{i+1,j}).$$

Tasks

5.1 Write `for` loops to move backwards through the vector array caculating the value of the option at each node.

5.2 Print out the tree to screen (with n=3) and compare to the simple example (click here, slide 19) to check your code is working.

5.3 If your values don't match - try to work out why!!

5.4 Now print out the value at $(0,0)$ increasing the number of steps in the tree. Do the results look feasible? Compare them against the exact values from the formula.

Finally the end of your code should look like this

CLICK HERE TO DOWNLOAD FULL CODE

```
1         }
2     }
3     // output the estimated option price
4     cout << " V(S="<<S0<<",t=0) = " << valueTree[0][0] << endl;
5     /** OUTPUT
6      V *(S=100,t=0) = 11.552
7      */
```

Tasks

5.5 Create a function returning the value of the binomial tree for a set of given parameters.

5.6 Write a code storing two time-levels, and compare (at every stage if needed) with the previous code.

5.7 Is it possible to store just one time-level? Try to write a code for this.

5.8 Do you notice any difference (time taken for computation) between the codes with different storage requirements?

5.3 Evaluating Convergence Properties

Taking the code from last time, we can finish this off by taking the main algorithm out into a function and allowing the payoff function to be set independently. A representative code might look like the following:

CLICK HERE TO DOWNLOAD FULL CODE

```
// return the payoff of the function you want to evaluate
double payoff(double S,double X)
{
    return max(S-X,0.);
}

// return the value of the binomial tree
double binomialTree(double S0,double X,double T,double r,double sigma,int n)
{
```

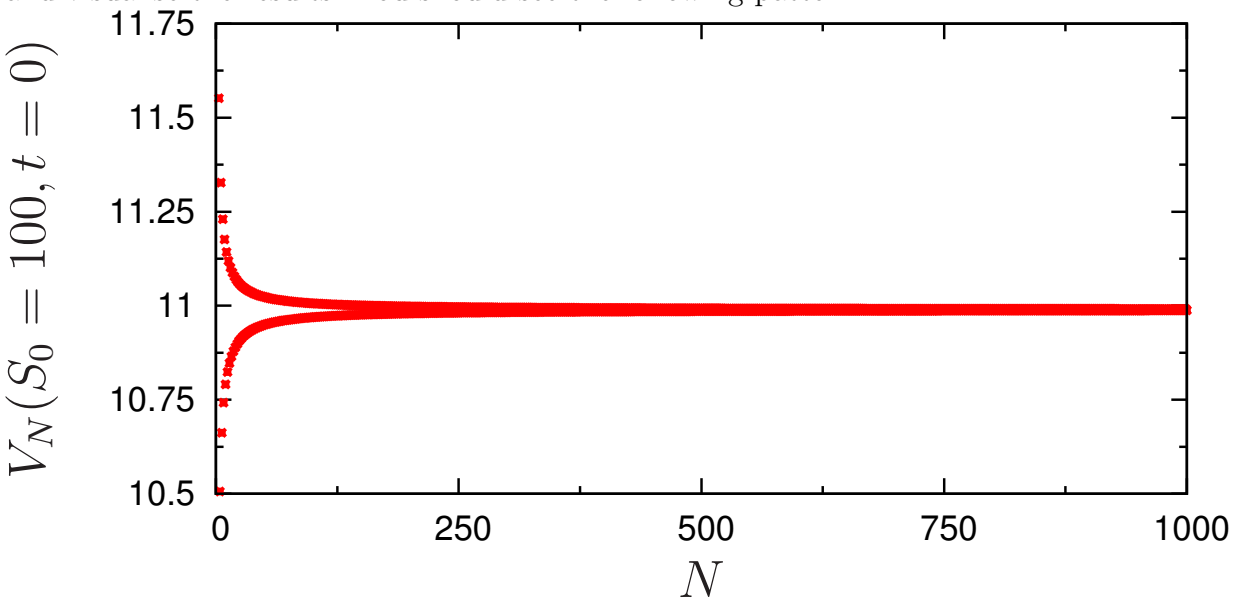
Now we wish to investigate what happens when N is increasing. Write the following loop in your code to output some results to file, you will need to adjust the output filename depending on you preference for where the file should be saved

CLICK HERE TO DOWNLOAD FULL CODE

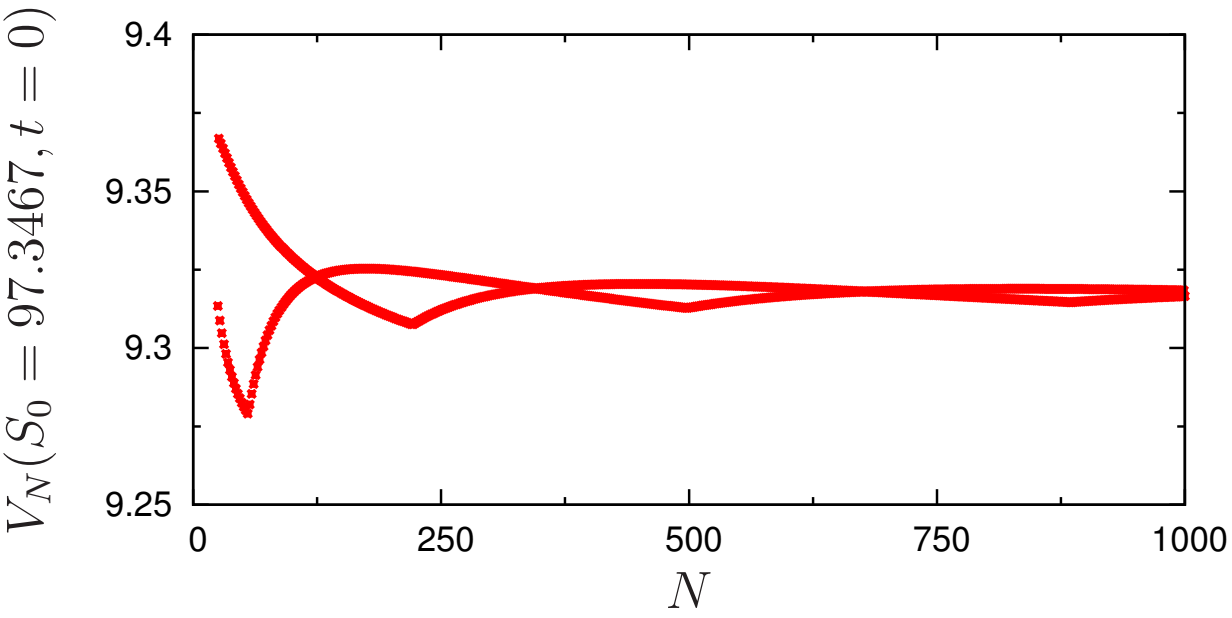
```
    // open output stream
    ofstream output("binomial-convergence.csv");
    // check it is open
    if(output.is_open())
    {
        cout << "File opened successfully" << endl;
        // output various n and V_n to file
        for(int n=3;n<=1000;n++)
        {
            output << n<<" , " << binomialTree(S0,X,T,r,sigma,n) << endl;
```

```
11          }
12          cout << "File write complete" << endl;
13      }
14      else
15      {
16          cout << "File could not be opened for some reason.\n";
17      }
18      /*
19       * OUTPUT
20       *  File opened successfully
21       *  File write complete
22       */
```

Run the code for $S_0 = 100$ and $X = 100$ and then import the data into a ploting package and visualise the results. You should see the following pattern



The pattern can be explained by realising that the nodes at terminal time will either be placed exactly on the strike price, or above/below depending on whether the number of steps is odd or even. We will see quite a different pattern if you choose $S_0 \neq X$, so for example set $S_0 = 97.3467$ in your code and rerun the results. After putting this in a plotting package you should see something like this



To produce this graph I have set $n = 25$ as the minimum grid size to show up the pattern better. This time we see *humps* as well as an odd-even effect. The humps are again caused by the positioning of the grid nodes relative to the strike price. Although the convergence of the tree is $O(1/n)$, the humps make it very difficult to achieve high accuracy. There are several papers that try to address this, including Leisen and Reimer (1996) and Heston and Zhou (2000), whilst Joshi (2007) gives a good comparison of different methods when American options are being priced.

Tasks

5.5 Run the code above and try to recreate the graphs.

5.6 Can you estimate the convergence rate?

5.7 Assume that the payoff of the option is

$$V(S,T) = \begin{cases} 0 & \text{if } S < X \\ 1 & \text{if } S \geq X \end{cases}$$

Plot out the solution to this problem with a binomial tree for different value s of N, in the case where $S_0 = X$ and $S_0 \neq X$. Can you explain the results?

5.8 Now try setting

$$V_{n,j} = \begin{cases} 0 & \text{if} \quad S_j^N < 2X - S_{j+1}^N \\ \frac{S_{j+1}^N + S_j^N - 2X}{S_{j+1}^N - S_{j-1}^N} & \text{if} \quad 2X - S_{j+1}^N \leq S_j^N \leq 2X - S_{j-1}^N \\ 1 & \text{if} \quad S_j^N > 2X - S_{j-1}^N \end{cases}$$

and plot the results again. How has it changed? See Heston and Zhou (2000) for more details on how to derive this formula.

References

Cox, J. C., Ross, S. A. and Rubinstein, M. (1979). Option pricing: A simplified approach, *Journal of financial Economics* **7**(3): 229–263.

Heston, S. and Zhou, G. (2000). On the rate of convergence of discrete-time contingent claims, *Mathematical Finance* **10**(1): 53–75.

Joshi, M. S. (2007). The convergence of binomial trees for pricing the american put, *Available at SSRN 1030143* .

Leisen, D. P. and Reimer, M. (1996). Binomial models for option valuation-examining and improving convergence, *Applied Mathematical Finance* **3**(4): 319–346.

Rendleman, R. J. and Bartter, B. J. (1979). Two-state option pricing, *The Journal of Finance* **34**(5): 1093–1110.